Instituto Tecnológico de Costa Rica
Escuela de Computadores
CE 1103: Algoritmos y Estructuras de Datos II

PROYECTO PROGRAMADO TRON

Estudiante
Guillermo Sánchez Cedeño

Profesor
Leonardo Araya Martinez

8 de septiembre, 2024

Índice General

1 INTRODUCCIÓN

Dentro del proyecto de TRON se nos presentan múltiples problemas destinados a la compresión y creación de estructuras de datos líneales en el lenguaje de progamación C#. Estas estructuras incluyen pilas, colas y listas simplemente enlazadas, donde tendremos que formarlas para que se adapten a los usos que les brindaremos en el manejo de ciertos items, poderes, o comportamientos dentro del juego.

1.1 Objetivo general

Desarrollar una solución al proyecto planteado utilizando las técnicas y algoritmos vistos en clase, aplicando de formas personalizadas la funcionalidad de estos para cumplir ciertos criterios críticos del funcionamiento de un juego similar a TRON clásico.

1.2 Objetivos específicos

- Desarrollar una lista simplemente enlazada para guardar datos posicionales de un jugador y su estela.
- Implementar una pila para almacenar 2 tipos de poderes, estos mismos se podrán ordenar según la directriz del jugador.
- Desarrollar una cola que almacene 3 tipos de objetos diferentes, cada uno de estos tendrá un efecto permanente en el comportamiento del jugador.
- Implementar los requerimientos anteriores mediante el uso de clases.
- Desarrollar un sistema de colisiones entre las estelas de distintos jugadores.
- Implementar bots que reaviven la inmersión del usuario.

2 BREVE DESCRIPCIÓN DEL PROBLEMA

2.1 Requerimientos

Dentro del proyecto se siguen varias reglas para desarrollar la solución, en primer lugar tenemos una matriz de nodos, cada uno de estos tendrá una referencia de los nodos adyacentes, es decir tendrán varios atributos llamados Up, Down, Left y Right, que harán una referencia a los objetos en esas direcciones.

Con ello podremos empezar a implementar nuestra primera estructura de datos, la lista simplemente enlazada que se verá reflejada en el juego como las motos. Esta estructura debe de tener una serie de atributos y métodos, como lo es su movimiento, el combustible, la detección de muerte del jugador o el manejo de la estela en conforme al movimiento.

Se debe de desarrollar una cola donde se almecenen objetos que son generados aleatoriamente en el mapa del juego, estos se aplicaran en un tiempo estimado que se verá en la interfaz del programa, así pues, serán el bidón de gasolina que recarga la gasolina del jugador, el objeto estela que aumenta en varias unidades la estela del jugador y por último la bomba que una vez ejecutada elimina al jugador.

A su vez la implementación de una pila que guarde poderes como la hiper velocidad que modifica la frecuencia de actualización de un jugador para imitar el efecto de mayor velocidad o el escudo que hace inmune al jugador de cualquier tipo de muerte excepto la bomba o la falta de gasolina. Estos poderes deben de ser ordenables por el jugador mediante el teclado.

2.2 Condiciones de juego

Las condiciones dentro del juego que desencadenaran múltiples eventos con el jugador o los bots son las siguientes:

- **Muerte de un jugador:** el jugador debe de morir cuando toque un nodo que contenga la estela o la cabeza de otra moto, en caso de tocar una cabeza la moto del jugador y la del bot deberán ser eliminadas. Además si el jugador se queda sin combustible deberá ser eliminado igualmente.

- **Activación de poderes:** el jugador debe de ordenar la pila de poderes mediante los números del teclado, una vez toque la tecla 1, el poder que se encuentre en la última posición de la pila será activado.
- **Condición de victoria:** el juego funciona con una cantidad prestablecida de bots, la misión del jugador es acabar con cada uno de ellos haciendoles pisar su estela destructiva, una vez se hayan acabado los bots dentro de la partida, el juego terminará con una interfaz de victoria.

3 DESCRIPCIÓN DE LA SOLUCIÓN

3.1 Requerimientos

- **Requerimiento 001:** La moto de un jugador singular dentro del juego ha sido implementada según las reglas que se siguen de la estructura de datos ”Lista Simplemente Enlazada”, donde esta posee un par de atributos, como lo es la *head*, que se refiere al primer elemento de la lista; en nuestra solución, esta cabeza será la referencia de la moto en el juego, y luego existen los nodos enlazados que formarán la estela destructiva que sigue a la moto.

  Los nodos siguen a su siguiente referencia tomando su posición anterior, es decir, cuando un nodo actualiza su posición en el espacio, deja una referencia de su posición anterior, esta será tomada por el siguiente nodo en la lista y así consecutivamente hasta llegar al último nodo.

- **Requerimiento 002:**

  - **Velocidad:** Para aplicar un valor aleatorio que determine la velocidad del jugador, se ha utilizado el método `Random.Range` de la clase `UnityEngine.Random`.
  - **Tamaño de la estela:** La clase de la moto almacena una variable de tipo entero llamada `size` que lleva la cuenta de cuántos nodos se han creado durante la ejecución del juego.
  - **Combustible:** Se ha desarrollado un contador que, cada cinco iteraciones del bucle encargado de actualizar la moto, resta una unidad al atributo `fuelRemaining` de la moto.

- **Cola de ítems:** Se ha implementado una cola jerárquica capaz de colocar los objetos de mayor importancia en la primera posición de esta, como lo serían los bidones de gasolina.
- **Pila de poderes:** Para desarrollar una pila que guarde los poderes, se optó por utilizar un modelo visto en clase, haciendo leves personalizaciones para los casos excepcionales del proyecto, como el dato que almacena la pila.

- **Requerimiento 003:** Se utilizó el algoritmo de creación de poderes y objetos dentro del juego, pero esta vez utilizando aquellos que se encuentran en la pila y cola de cada moto.

- **Requerimiento 004:** El jugador, con los botones del teclado, podrá colocar uno de sus poderes en el tope de la pila. Esto se implementó por medio de una cola jerárquica auxiliar que encola los objetos que estaban antes del objeto seleccionado. Este se guarda en una variable y, posteriormente, se vuelven a apilar los objetos dentro de la cola. Por último, el objeto seleccionado se apila de primero.

- **Requerimiento 005:** Los objetos se aplicarán mediante el uso de una cola jerárquica, donde los bidones de gasolina tienen una prioridad de 0, siendo esta la más alta. Por otro lado, están los objetos estela y la bomba, ambos con una prioridad de 1 dentro de la pila.

  En pantalla se muestra una cuenta atrás de 7 segundos que indica cuándo se aplicará el objeto en la primera posición.

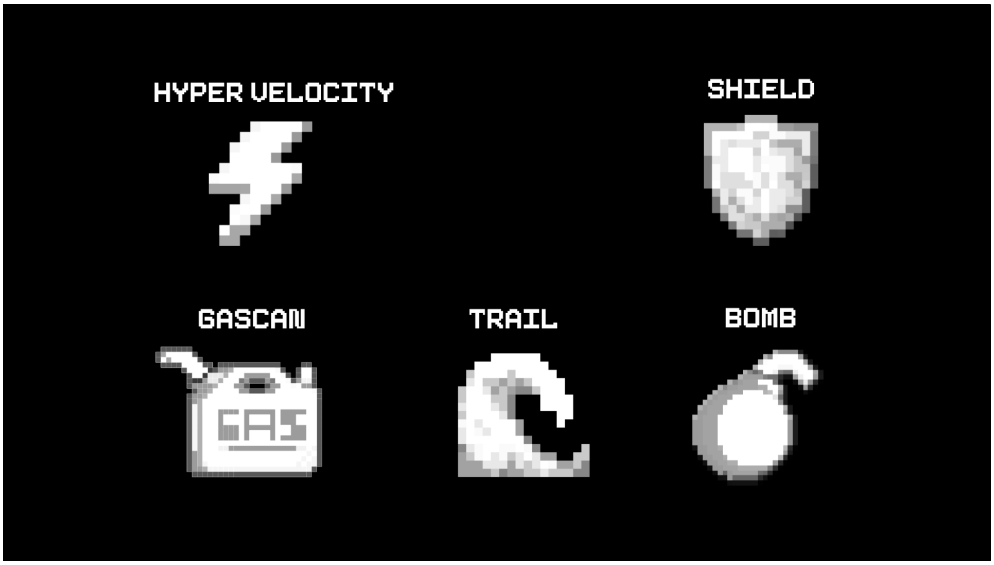
- **Requerimiento 006:** Cada nodo dentro de la matriz tiene un estado, estos varían en función de los eventos del juego. Puede ser que el nodo tenga estado desocupado, es decir, no hay nada ocupando su espacio, o puede tener estado de estela, lo cual ocurre cuando la estela destructiva de una moto se mueve por el nodo. Con esto, podemos hacer una validación por cada actualización de una moto, y si esta pisa un nodo con estado estela o cabeza, se desencadenará el evento correspondiente.

- **Requerimiento 007:** Se ha creado una matriz de nodos. Como se comentó antes, cada nodo tiene cuatro atributos separados llamados `Up`, `Down`, `Left` y `Right`, que hacen referencia a sus nodos adyacentes. Gracias a esta red podremos facilitar el movimiento del jugador, pues, por cada actualización, lo moveremos a la posición del siguiente nodo, conforme a la dirección que lleve.

- **Requerimiento 008:**

- **Generación de objetos:** Cada 5 segundos, en la malla de nodos se genera automáticamente un objeto, siguiendo ciertas validaciones que impiden generarlo dentro de un nodo que previamente tiene otro objeto dentro.
- **Combustible:** El ítem se representa como un bidón de gasolina dentro del juego. Una vez recogido y ejecutado por el jugador, calculará un número del 15 al 25 y se sumará al atributo `fuelRemaining` de la moto correspondiente.
- **Crecimiento de la estela:** El ítem se representa como una ola dentro del juego. Este escoge un valor del 1 al 10 y agrega a la estela utilizando el método `addTrail` de cada instancia de una moto.

  El método `addTrail` itera sobre toda la lista y, una vez llegado al final de esta, calcula la dirección en la que se deben generar los nodos (esta dirección debe ser contraria a la dirección del último nodo).
- **Bombas:** Una vez recogidas y ejecutadas, estas ejecutan el método `dieEvent` de las motos, que borra su estela, los *GameObjects* asociados a esta y la cabeza.
- **Escudo:** Cada una de las instancias de las motos tiene el atributo booleano `isInmune`. Si este es `false`, los eventos que desencadenan la muerte de una moto se desarrollarán según lo previsto. En caso contrario, todas estas verificaciones serán saltadas y el jugador podrá ser invulnerable por 5 segundos.
- **Hiper Velocidad:** Cada una de las motos tiene un atributo `velocidad` asociado, que indica cada cuánto se deben actualizar. Así, al cambiar este atributo a uno más bajo, se genera un aumento de velocidad. Esta es la lógica que sigue este poder, que tiene una duración total de 5 segundos y no drenará combustible mientras esté activo.



- **Bots:** Los bots tienen un comportamiento parecido al de los jugadores, solo que estos eligen el camino a seguir mediante una serie de validaciones que se guardan en una lista. Si el bot decide hacer un movimiento, usará la casilla en la que se encuentra actualmente y evaluará los siguientes movimientos conforme a las siguientes preguntas: ¿Es esta dirección contraria a mi dirección actual? ¿Alguno de los nodos adyacentes es nulo? ¿Los estados de los nodos son diferentes a desocupado? Si lo es, no se agrega a la lista de posibles movimientos.

  Además, los bots no podrán chocar con los límites del mapa, pues estos detectarán que su próximo movimiento es un nodo nulo y, por ende, cambiarán su dirección a una más segura.

Diagrama UML